In [10]:

```
def recAOStar(n):
    global finalPath
    print("Expanding Node:",n)
    and_nodes=[]
    or_nodes=[]
    if(n in allNodes):
        if 'AND' in allNodes[n]:
            and_nodes=allNodes[n]['AND']
        if 'OR' in allNodes[n]:
            or_nodes=allNodes[n]['OR']
    if len(and_nodes)==0 and len(or_nodes)==0:
        return
    solvable=False
    marked={}
    while not solvable:
        if len(marked)==len(and_nodes)+len(or_nodes):
            min_cost_least,min_cost_group_least=least_cost_group(and_nodes,or_nodes,{})
            solvable=True
            change_heuristic(n,min_cost_least)
            optimal_child_group[n]=min_cost_group_least
            continue
        min_cost,min_cost_group=least_cost_group(and_nodes,or_nodes,marked)
        is_expanded=False
        if len(min_cost_group)>1:
            if(min_cost_group[0] in allNodes):
                is_expanded=True
                recAOStar(min_cost_group[0])
            if(min_cost_group[1] in allNodes):
                is_expanded=True
                recAOStar(min_cost_group[1])
        else:
            if(min_cost_group in allNodes):
                is_expanded=True
                recAOStar(min_cost_group)
        if is_expanded:
            min_cost_verify,min_cost_group_verify=least_cost_group(and_nodes,or_nodes,{})
            if min_cost_group==min_cost_group_verify:
                solvable=True
                change_heuristic(n,min_cost_verify)
                optimal_child_group[n]=min_cost_group
        else:
            solvable=True
            change_heuristic(n,min_cost)
            optimal_child_group[n]=min_cost_group
        marked[min_cost_group]=1
    return heuristic(n)
def least_cost_group(and_nodes,or_nodes,marked):
    node_wise_cost={}
    for node_pair in and_nodes:
        if not node_pair[0]+node_pair[1] in marked:
            cost=0
            cost=cost+heuristic(node_pair[0])+heuristic(node_pair[1])+2
            node_wise_cost[node_pair[0]+node_pair[1]]=cost
    for node in or_nodes:
        if not node in marked:
            cost=0
            cost=cost+heuristic(node)+1
            node_wise_cost[node]=cost
    min_cost=999999
    min_cost_group=None
    for costKey in node_wise_cost:
        if node_wise_cost[costKey]<min_cost:
            min_cost=node_wise_cost[costKey]
            min_cost_group=costKey
    return [min_cost,min_cost_group]
def heuristic(n):
    return H_dist[n]
def change_heuristic(n,cost):
    H_dist[n]=cost
    return
def print_path(node):
    print(optimal_child_group[node],end="")
    node=optimal_child_group[node]
    if len(node)>1:
        if node[0] in optimal_child_group:
            print("->",end="")
            print_path(node[0])
        if node[1] in optimal_child_group:
            print("->",end="")
            print_path(node[1])
    else:
        if node in optimal_child_group:
            print("->",end="")
            print_path(node)
H_dist={
    'A':-1,
    'B':[illegible],
    'C':[illegible],
    'D':[illegible],
    'E':[illegible],
    'F':[illegible],
    'G':[illegible],
    'H':[illegible],
    'I':[illegible],
    'J':[illegible]
}
allNodes={
    'A':{'AND':[('C','D')],'OR':['B']},
    'B':{'OR':['E','F']},
    'C':{'OR':['G'],'AND':[('H','I')]},
    'D':{'OR':['J']}
}
optimal_child_group={}
optimal_cost=recAOStar('A')
print('Nodes which gives optimal cost are')
print_path('A')
print('\nOptimal cost is::',optimal_cost)
```

```
Expanding Node: A
Expanding Node: B
Expanding Node: C
Expanding Node: D
Nodes which gives optimal cost are
CD->HI->J
Optimal cost is:: 5
```

In [ ]:

In [1]:

```python
def recAOStar(n):
    global finalPath
    print("Expanding Node:",n)
    and_nodes=[]
    or_nodes=[]
    if(n in allNodes):
        if 'AND' in allNodes[n]:
            and_nodes=allNodes[n]['AND']
        if 'OR' in allNodes[n]:
            or_nodes=allNodes[n]['OR']
    if len(and_nodes)==0 and len(or_nodes)==0:
        return
    solvable=False
    marked={}
    while not solvable:
        if len(marked)==len(and_nodes)+len(or_nodes):
            min_cost_least,min_cost_group_least=least_cost_group(and_nodes,or_nodes,{})
            solvable=True
            change_heuristic(n,min_cost_least)
            optimal_child_group[n]=min_cost_group_least
            continue
        min_cost,min_cost_group=least_cost_group(and_nodes,or_nodes,marked)
        is_expanded=False
        if len(min_cost_group)>1:
            if(min_cost_group[0] in allNodes):
```

```
            if(min_cost_group[0] in allNodes):
                is_expanded=True
                recAOStar(min_cost_group[0])
            if(min_cost_group[1] in allNodes):
                is_expanded=True
                recAOStar(min_cost_group[1])
        else:
            if(min_cost_group in allNodes):
                is_expanded=True
                recAOStar(min_cost_group)
        if is_expanded:
            min_cost_verify,min_cost_group_verify=least_cost_group(and_nodes,or_nodes,{})
            if min_cost_group==min_cost_group_verify:
                solvable=True
                change_heuristic(n,min_cost_verify)
                optimal_child_group[n]=min_cost_group
        else:
            solvable=True
            change_heuristic(n,min_cost)
            optimal_child_group[n]=min_cost_group
        marked[min_cost_group]=1
    return heuristic(n)
def least_cost_group(and_nodes,or_nodes,marked):
    node_wise_cost={}
    for node_pair in and_nodes:
        if not node_pair[0]+node_pair[1] in marked:
            cost=0
```

```
            cost=0
            cost=cost+heuristic(node_pair[0])+heuristic(node_pair[1])+2
            node_wise_cost[node_pair[0]+node_pair[1]]=cost
    for node in or_nodes:
        if not node is marked:
            cost=0
            cost=cost+heuristic(node)+1
            node_wise_cost[node]=cost
    min_cost=999999
    min_cost_group=None
    for costKey in node_wise_cost:
        if node_wise_cost[costKey]<min_cost:
            min_cost=node_wise_cost[costKey]
            min_cost_group=costKey
    return [min_cost,min_cost_group]
def heuristic(n):
    return H_dist[n]
def change_heuristic(n,cost):
    H_dist[n]=cost
    return
def print_path(node):
    print(optimal_child_group[node],end="")
    node=optimal_child_group[node]
    if len(node)>1:
        if node[0] in optimal_child_group:
            print("->",end="")
            print_path(node[0])
        if node[1] in optimal_child_group:
```

```
        if node[1] in optimal_child_group:
            print("->",end="")
            print_path(node[1])
    else:
        if node in optimal_child_group:
            print("->",end="")
            print_path(node)
H_dist={
    'A':-1,
    'B':4,
    'C':2,
    'D':3,
    'E':6,
    'F':8,
    'G':2,
    'H':0,
    'I':0,
    'J':0
}
allNodes={
    'A':{'AND':[('C','D')],'OR':['B']},
    'B':{'OR':['E','F']},
    'C':{'OR':['G'],'AND':[('H','I')]},
    'D':{'OR':['J']}
}
optimal_child_group={}
optimal_cost=recAOStar('A')
```

```
 99     'B':{'OR':['E','F']},
100     'C':{'OR':['G'],'AND':[('H','I')]},
101     'D':{'OR':['J']}
102 }
103 optimal_child_group={}
104 optimal_cost=recAOStar('A')
105 print('Nodes which gives optimal cost are')
106 print_path('A')
107 print('\nOptimal cost is::',optimal_cost)
```

```
Expanding Node: A
Expanding Node: B
Expanding Node: C
Expanding Node: D
Nodes which gives optimal cost are
CD->HI->J
Optimal cost is:: 5
```

In [ ]: 1